Supervised classification for driver types with aggregation analysis.

In [ ]:
```
import pandas as pd
```

In [2]:
```
fcd_df = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/fcd-output
emission_df = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/emiss:
features = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/features
```

In [28]:
```
features
```

Out[28]:

	Unnamed: 0	time	id	x	y	angle	type	speed	pos	lane
0	0	0.0	f_0.0	-38.09	-4.80	90.00	SportsCarDriver	0.00	61.91	E0_0
1	1	0.1	f_0.0	-38.08	-4.80	90.00	SportsCarDriver	0.03	61.92	E0_0
2	2	0.2	f_0.0	-38.07	-4.80	90.00	SportsCarDriver	0.10	61.93	E0_0
3	3	0.3	f_0.0	-38.05	-4.80	90.00	SportsCarDriver	0.21	61.95	E0_0
4	4	0.4	f_0.0	-38.01	-4.80	90.00	SportsCarDriver	0.40	61.99	E0_0
...	...	...	...	...	...	...	...	...	...	...
398475	398475	444.3	f_5.99	87.30	10.02	294.06	CautiousDriver	16.62	5.14	:J2_0_0
398476	398476	444.4	f_5.99	85.91	10.87	300.53	CautiousDriver	16.55	6.79	:J2_0_0
398477	398477	444.5	f_5.99	84.29	11.17	296.05	CautiousDriver	16.48	8.44	:J2_0_0
398478	398478	444.6	f_5.99	82.66	11.27	285.32	CautiousDriver	16.41	1.02	E8_0
398479	398479	444.7	f_5.99	81.03	11.25	274.53	CautiousDriver	16.34	2.65	E8_0

398480 rows × 16 columns

In [29]:
```
# Merging the data using 'id' and 'type'
merged_data = pd.merge(fcd_df,emission_df, on=['time','id','type'])
```

In [30]:
```
# Merging the data using 'id' and 'type'
merged_data = pd.merge(merged_data,features, on=['time','id','type'])
```

In [31]:
```
merged_data
```

Out[31]:

	time	id	x_x	y_x	angle_x	type	speed_x	pos_x	la
0	0.0	f_0.0	-38.09	-4.80	90.00	SportsCarDriver	0.00	61.91	
1	0.0	f_1.0	-57.03	-37.67	33.37	CautiousDriver	0.00	75.68	
2	0.0	f_2.0	-33.33	-1.60	90.00	ElderlyDriver	0.00	66.67	
3	0.0	f_3.0	357.14	4.80	270.00	AggressiveLanechangingdriver	0.00	42.86	
4	0.0	f_4.0	241.15	21.32	215.86	AggressiveLanechangingdriver	0.00	98.24	
...	...	...	...	...	...	...	...	...	
398475	999.9	f_5.222	111.69	1.60	270.00	SportsCarDriver	0.00	102.38	
398476	999.9	f_5.223	245.35	4.80	270.00	InexperiencedDriver	12.57	55.00	
398477	999.9	f_5.224	158.14	4.80	270.00	SportsCarDriver	26.32	55.93	
398478	999.9	f_5.225	335.19	1.60	270.00	ElderlyDriver	8.05	64.81	
398479	999.9	f_5.226	367.06	4.80	270.00	ElderlyDriver	0.04	32.94	

398480 rows × 41 columns

In [32]:
```
final = merged_data.drop(columns=['slope','eclass','route','waiting','lane_y'
```

In [33]:
```
final
```

Out[33]:

	time	id	x_x	y_x	angle_x	type	speed_x	pos_x	a
0	0.0	f_0.0	-38.09	-4.80	90.00	SportsCarDriver	0.00	61.91	
1	0.0	f_1.0	-57.03	-37.67	33.37	CautiousDriver	0.00	75.68	
2	0.0	f_2.0	-33.33	-1.60	90.00	ElderlyDriver	0.00	66.67	
3	0.0	f_3.0	357.14	4.80	270.00	AggressiveLanechangingdriver	0.00	42.86	
4	0.0	f_4.0	241.15	21.32	215.86	AggressiveLanechangingdriver	0.00	98.24	
...	...	...	...	...	...	...	...	...	
398475	999.9	f_5.222	111.69	1.60	270.00	SportsCarDriver	0.00	102.38	
398476	999.9	f_5.223	245.35	4.80	270.00	InexperiencedDriver	12.57	55.00	
398477	999.9	f_5.224	158.14	4.80	270.00	SportsCarDriver	26.32	55.93	
398478	999.9	f_5.225	335.19	1.60	270.00	ElderlyDriver	8.05	64.81	
398479	999.9	f_5.226	367.06	4.80	270.00	ElderlyDriver	0.04	32.94	

398480 rows × 17 columns

In [34]:
```
from sklearn.preprocessing import LabelEncoder
# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode categorical columns
final['type'] = label_encoder.fit_transform(final['type'])
final['id'] = label_encoder.fit_transform(final['id'])
```

In [35]:
```
final.corr()
```

Out[35]:

	time	id	x_x	y_x	angle_x	type	sp
time	1.000000	-0.027918	-0.011356	0.005042	0.014711	-0.014541	-0.0
id	-0.027918	1.000000	0.327327	0.409731	0.838318	0.065246	-0.2
x_x	-0.011356	0.327327	1.000000	0.386571	0.391870	0.017162	0.0
y_x	0.005042	0.409731	0.386571	1.000000	0.518102	0.037787	0.0
angle_x	0.014711	0.838318	0.391870	0.518102	1.000000	0.061652	-0.0
type	-0.014541	0.065246	0.017162	0.037787	0.061652	1.000000	0.0
speed_x	-0.047372	-0.203942	0.076041	0.001827	-0.078379	0.039832	1.0
pos_x	0.032194	0.066977	-0.114310	0.007374	-0.005626	0.013670	-0.1
acceleration_x	-0.021207	-0.047929	-0.026528	-0.005661	-0.045198	0.006287	-0.1
CO	-0.007374	0.047427	-0.039143	-0.006916	0.017719	0.085217	-0.3
fuel	-0.032181	-0.061821	-0.019635	0.002178	-0.014780	0.090033	0.2
noise	-0.060527	-0.173792	0.054022	0.003849	-0.077087	0.085638	0.6
angle	0.014711	0.838318	0.391870	0.518102	1.000000	0.061652	-0.0
lane_length	0.015000	0.036952	-0.226934	-0.112893	-0.090701	0.000577	-0.1
lane_change	-0.003851	-0.017214	0.012718	0.002876	0.000320	0.004922	0.0
vehicle_density	-0.004640	0.017209	0.014581	-0.016119	0.016353	-0.004523	-0.0
avg_speed_nearby_vehicles	-0.050949	-0.214982	0.067308	-0.013411	-0.087862	0.029756	0.9

Merging Mean,Variance and Standard Deviation data

In [36]:
```
import pandas as pd

# Assuming 'df' contains the provided DataFrame

# Group by 'id' and 'time' and calculate the mean, std, and var for each colum
mean_values = final.groupby(['id', 'type']).mean().reset_index()
std_values = final.groupby(['id', 'type']).std().reset_index()
var_values = final.groupby(['id', 'type']).var().reset_index()

# Merge the mean, std, and var DataFrames
result = pd.merge(mean_values, std_values, on=['id', 'type'], suffixes=('_mean
result = pd.merge(result, var_values, on=['id', 'type'], suffixes=('', '_var')

# Display the resulting DataFrame with mean, std, and var for each column
print(result)
```

```
        id  type  time_mean    x_x_mean   y_x_mean  angle_x_mean  \
0        0     4       7.50  107.577550  -4.800000     90.000000   
1        1     2      16.00   87.693980  -3.451505     90.000000   
2        2     4      37.95  140.579252  -4.201869     90.000000   
3        3     1     364.15   74.287206  -4.800000     90.000000   
4        4     2     368.30   85.125269  -1.600000     90.000000   
...    ...   ...        ...         ...        ...           ...   
1266  1266     3     417.75  232.513088   7.357618    275.632353   
1267  1267     3     421.75  227.060992   5.506031    271.744313   
1268  1268     1     432.75  248.253301  10.328058    277.318204   
1269  1269     2     430.50  206.437138  10.499731    278.446229   
1270  1270     1     432.35  250.277097   5.909798    270.869234   

      speed_x_mean  pos_x_mean  acceleration_x_mean     CO_mean  ...  \
0        23.237682   47.523974             1.463444  240.779139  ...   
1        12.496923   47.752943             0.657291   59.650535  ...   
2        20.085093   45.981869             0.996636  160.531729  ...   
3        12.905353   42.313765             0.627265   65.949529  ...   
4        12.671254   48.519319             0.670789   59.147742  ...   
...            ...         ...                  ...         ...  ...   
1266     10.433412   43.509000             0.375382   60.537941  ...   
1267     10.203282   48.202863             0.555420   60.487786  ...   
1268      9.568932   44.238835             0.351092   58.010413  ...   
1269     11.411111   50.160101             0.534141   62.723064  ...   
1270     10.555524   50.547944             0.659032   68.910524  ...   

            pos_x  acceleration_x            CO       fuel      noise  \
0      949.568609       10.203543  41517.428541  23.190705  45.237318   
1      879.061975        0.211585   2212.185192   0.247460   9.977822   
2      838.181464        8.112813  29855.731260  15.921622  49.947717   
3      731.332891        0.185713   1779.774978   0.620957  14.549115   
4      886.990330        0.215334   2186.829440   0.219426   6.459293   
...           ...             ...           ...        ...        ...   
1266   999.195548        1.002605   2031.932277   0.586912   8.121168   
1267  1201.809459        0.220230   2524.072358   0.308355   4.932657   
1268   824.095973        0.501527   2413.930779   0.651597  13.694340   
1269   999.236114        0.954994   1891.470833   0.476405   5.947707   
1270   963.310085        0.228923   2202.294516   0.403851   7.112195   

           angle  lane_length  lane_change  vehicle_density  \
0       0.000000   904.924682     0.056424         0.001447   
1       0.000000   773.581594     0.035555         0.001277   
2       0.000000   701.116068     0.040477         0.001276   
3       0.000000   639.056340     0.025846         0.001084   
4       0.000000   808.739493     0.031330         0.001287   
...          ...          ...          ...              ...   
1266  230.048384   962.355210     0.025846         0.000562   
1267   55.316171  1216.862500     0.033298         0.000967   
1268  310.465477   822.919064     0.019087         0.000535   
1269  346.527782  1123.479815     0.029484         0.000749   
1270   22.829946  1063.800649     0.027540         0.000791   

      avg_speed_nearby_vehicles  
0                    113.383921  
1                     39.974584  
2                     51.891929  
3                     53.400125  
```

```
4                      37.797974
...                          ...
1266                   18.254119
1267                   24.101123
1268                   25.566286
1269                   21.923327
1270                   22.718884

[1271 rows x 47 columns]
```

In [37]:
```
result.fillna(value=0, inplace=True)  # Replaces NaN with 0
```

In [38]:
```
result.columns
```

Out[38]:
```
Index(['id', 'type', 'time_mean', 'x_x_mean', 'y_x_mean', 'angle_x_mean',
       'speed_x_mean', 'pos_x_mean', 'acceleration_x_mean', 'CO_mean',
       'fuel_mean', 'noise_mean', 'angle_mean', 'lane_length_mean',
       'lane_change_mean', 'vehicle_density_mean',
       'avg_speed_nearby_vehicles_mean', 'time_std', 'x_x_std', 'y_x_std',
       'angle_x_std', 'speed_x_std', 'pos_x_std', 'acceleration_x_std',
       'CO_std', 'fuel_std', 'noise_std', 'angle_std', 'lane_length_std',
       'lane_change_std', 'vehicle_density_std',
       'avg_speed_nearby_vehicles_std', 'time', 'x_x', 'y_x', 'angle_x',
       'speed_x', 'pos_x', 'acceleration_x', 'CO', 'fuel', 'noise', 'angle',
       'lane_length', 'lane_change', 'vehicle_density',
       'avg_speed_nearby_vehicles'],
      dtype='object')
```

In [52]:
```
result['type'].value_counts()
```

Out[52]:
```
type
3    406
2    265
4    249
1    239
0    112
Name: count, dtype: int64
```

Random Forest(ensemble Method)

In [39]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Assuming 'df' is your DataFrame containing the data

# Features (X) and target variable (y)
X = result.drop(['type','id'],axis=1)  # Features
y = result['type']  # Target variable

# Split the data into training and testing sets (adjust test_size and random_s
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand

# Initialize the classifier (you can use any other classifier of your choice)
clf_rand = RandomForestClassifier(n_estimators=200, random_state=42)

# Fit the classifier on the training data
clf_rand.fit(X_train, y_train)

# Predict on the test data
y_pred = clf_rand.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.7450980392156863
```

KNN-Model

In [40]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score



# Features (X) and target variable (y)
X = result.drop(['type','id'], axis=1)  # Features
y = result['type']  # Target variable


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize the k-NN classifier
k = 5  # You can change this value as needed
knn = KNeighborsClassifier(n_neighbors=k)

# Fit the model on the training data
knn.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = knn.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the k-NN model: {accuracy:.2f}')
```

```
Accuracy of the k-NN model: 0.45
```

DNN Model

In [41]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import regularizers
from tensorflow.keras.utils import to_categorical

# Assuming 'fcd_df' is your DataFrame containing the data

# Features (X) and target variable (y)
X = result.drop(['type','id'], axis=1)  # Features
y = result['type']  # Target variable

# Convert y to categorical (one-hot encoded)
y = to_categorical(y)
num_classes = 5
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define the Deep Neural Network model with L2 regularization and dropout
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu', kernel_re
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.0
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.0
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))  # For multi-class classi

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac

# Fit the model on the training data
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test

# Evaluate the model on the test data
_, accuracy = model.evaluate(X_test, y_test)
print(f"Accuracy: {accuracy}")
```

```
Epoch 1/10
32/32 [==============================] - 1s 7ms/step - loss: 3.7146 - accura
cy: 0.2549 - val_loss: 3.2458 - val_accuracy: 0.4667
Epoch 2/10
32/32 [==============================] - 0s 3ms/step - loss: 3.2216 - accura
cy: 0.3947 - val_loss: 2.9941 - val_accuracy: 0.4902
Epoch 3/10
32/32 [==============================] - 0s 3ms/step - loss: 2.9504 - accura
cy: 0.4341 - val_loss: 2.7425 - val_accuracy: 0.4902
Epoch 4/10
32/32 [==============================] - 0s 3ms/step - loss: 2.7408 - accura
cy: 0.4675 - val_loss: 2.5296 - val_accuracy: 0.4902
Epoch 5/10
32/32 [==============================] - 0s 3ms/step - loss: 2.5788 - accura
cy: 0.4528 - val_loss: 2.3764 - val_accuracy: 0.4902
Epoch 6/10
32/32 [==============================] - 0s 3ms/step - loss: 2.4209 - accura
cy: 0.4587 - val_loss: 2.2328 - val_accuracy: 0.4902
Epoch 7/10
32/32 [==============================] - 0s 3ms/step - loss: 2.2669 - accura
cy: 0.4547 - val_loss: 2.1011 - val_accuracy: 0.4902
Epoch 8/10
32/32 [==============================] - 0s 2ms/step - loss: 2.1251 - accura
cy: 0.4715 - val_loss: 1.9885 - val_accuracy: 0.4902
Epoch 9/10
32/32 [==============================] - 0s 3ms/step - loss: 2.0118 - accura
cy: 0.4862 - val_loss: 1.8888 - val_accuracy: 0.4902
Epoch 10/10
32/32 [==============================] - 0s 3ms/step - loss: 1.8991 - accura
cy: 0.4902 - val_loss: 1.8035 - val_accuracy: 0.4902
8/8 [==============================] - 0s 2ms/step - loss: 1.8035 - accurac
y: 0.4902
Accuracy: 0.4901960790157318
```

Gaussian Naive Bayes

In [42]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Assuming 'fcd_df' is your DataFrame containing the data

# Features (X) and target variable (y)
X = result.drop(['type','id'], axis=1)  # Features
y = result['type']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand

# Initialize the MinMaxScaler and fit-transform the training data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)

# Transform the test data using the same scaler
X_test = scaler.transform(X_test)

# Initialize the Gaussian Naive Bayes classifier
clf = GaussianNB()

# Fit the classifier on the training data
clf.fit(X_train, y_train)

# Predict on the test data
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.6039215686274509
```

Best Model Performance on unseen data(New Simulation data)

In [19]:
```
fcd_df_New = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/fcd-ou
emission_df_New = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/e
features_New = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/feat
```

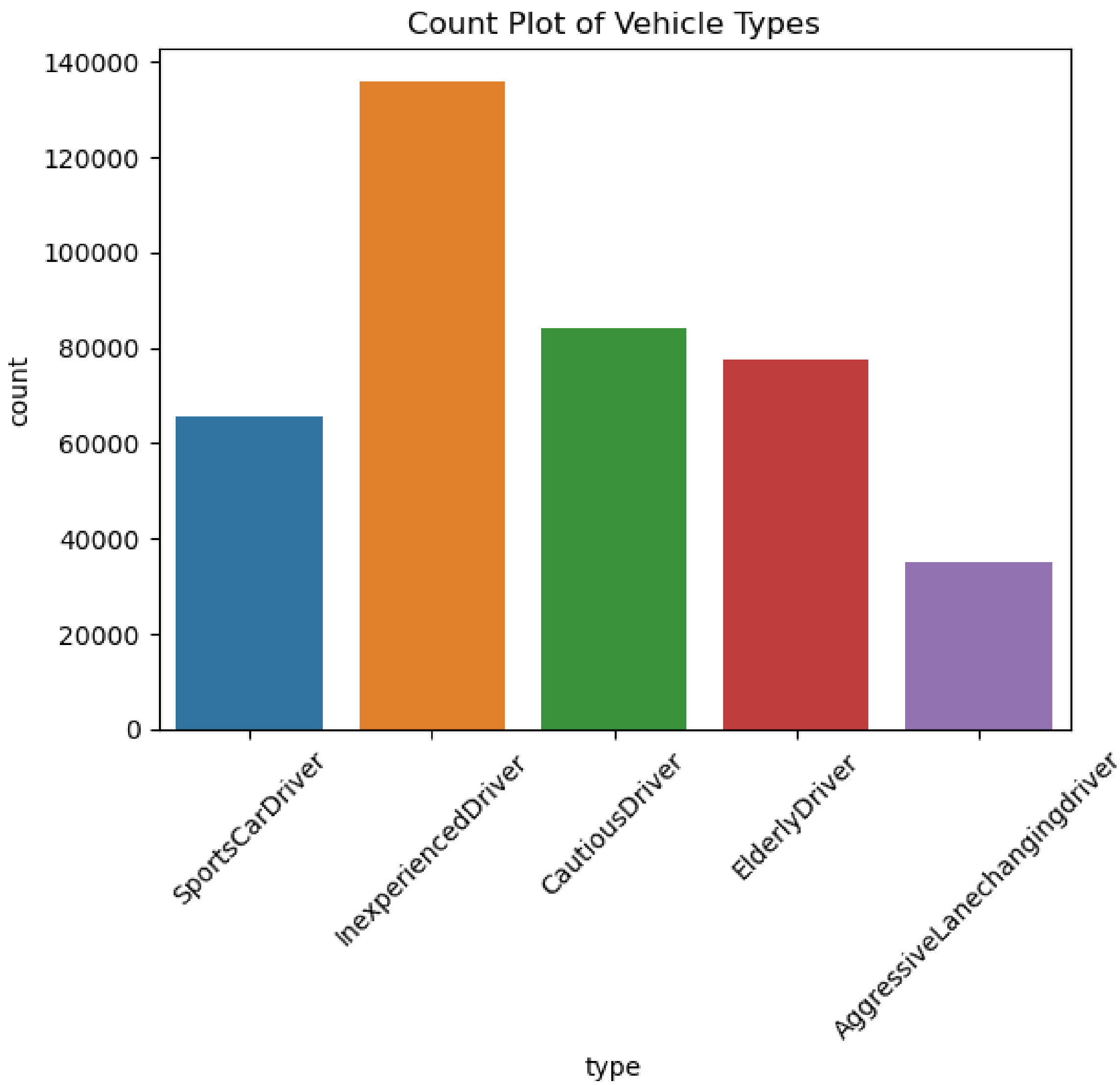
In [43]:
```
# Merging the data using 'id' and 'type'
merged_data_New = pd.merge(fcd_df_New,emission_df_New, on=['time','id','type'
```

In [44]:
```
# Merging the data using 'id' and 'type'
merged_data_New = pd.merge(merged_data_New,features_New, on=['time','id','type
```

Count Plot of Vehicle Types

In [55]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Example of creating a count plot for the 'type' column
sns.countplot(data=merged_data_New, x='type')
plt.title('Count Plot of Vehicle Types')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if need
plt.show()
```



In [45]:
```
final_New = merged_data_New.drop(columns=['slope','eclass','route','waiting',
```

In [46]:
```
from sklearn.preprocessing import LabelEncoder
# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode categorical columns
final_New['type'] = label_encoder.fit_transform(final_New['type'])
final_New['id'] = label_encoder.fit_transform(final_New['id'])
```

In [47]:
```
import pandas as pd

# Assuming 'df' contains the provided DataFrame

# Group by 'id' and 'time' and calculate the mean, std, and var for each colum
mean_values = final_New.groupby(['id', 'type']).mean().reset_index()
std_values = final_New.groupby(['id', 'type']).std().reset_index()
var_values = final_New.groupby(['id', 'type']).var().reset_index()

# Merge the mean, std, and var DataFrames
result_New = pd.merge(mean_values, std_values, on=['id', 'type'], suffixes=('_
result_New = pd.merge(result_New, var_values, on=['id', 'type'], suffixes=(''

# Display the resulting DataFrame with mean, std, and var for each column
print(result_New)
```

```
        id  type  time_mean    x_x_mean   y_x_mean  angle_x_mean  \
0        0     4       9.00  142.739171  -4.800000     90.000000   
1        1     3      17.95   52.987160  -3.209467     90.000000   
2        2     1      51.55  108.559641  -3.774586     90.000000   
3        3     2     365.00   80.617143  -4.800000     90.000000   
4        4     2     376.70   54.816736  -1.600000     90.000000   
...    ...   ...        ...         ...        ...           ...   
1267  1267     2     416.55  270.367073   5.593496    270.000000   
1268  1268     1     424.45  256.381689   4.590338    271.160473   
1269  1269     2     436.25  216.555180  11.507335    278.466437   
1270  1270     2     436.35  225.095988   6.357685    276.368056   
1271  1271     3     434.35  229.291532   6.304073    271.350685   

      speed_x_mean  pos_x_mean  acceleration_x_mean     CO_mean  ...  \
0        20.700552   44.590884             1.043204  145.199448  ...   
1        12.304763   37.877456             0.543343   64.902781  ...   
2        12.019586   44.773785             0.387403   52.506436  ...   
3        12.947944   46.996620             0.697666   62.435331  ...   
4        11.289034   39.785326             0.503812   56.720209  ...   
...            ...         ...                  ...         ...  ...   
1267     10.962724   46.412114             0.176992   63.994268  ...   
1268     10.374392   44.283986             0.362128   56.202095  ...   
1269     10.652605   48.867754             0.454311   56.270659  ...   
1270     10.213364   50.772130             0.485062   64.418981  ...   
1271     10.429556   48.939435             0.550726   67.486129  ...   

            pos_x  acceleration_x            CO       fuel      noise  \
0     1019.697413        6.982777  19547.554932  10.832321  35.822783   
1      846.849002        0.276305   1554.719180   0.635933  11.356579   
2      734.573604        0.495733   2030.685574   0.497010   6.973152   
3      819.476962        0.251244   2162.356078   0.387623   7.454172   
4      899.662123        0.486971   2198.392627   0.408777  16.535861   
...           ...             ...           ...        ...        ...   
1267   809.955173        3.044184   2325.792560   0.646991  15.003130   
1268   946.908198        0.516791   2389.770224   0.634483   5.678417   
1269   909.139919        0.272772   2018.952948   0.477212   6.751585   
1270   912.561277        0.684037   2215.847815   0.527766  11.589522   
1271  1231.874856        0.451544   2336.980617   0.814688   6.293530   

           angle  lane_length  lane_change  vehicle_density  \
0       0.000000   884.953263     0.047514         0.001546   
1       0.000000   649.869537     0.028796         0.001054   
2       0.000000   651.669529     0.026936         0.001087   
3       0.000000   779.688419     0.030482         0.001084   
4       0.000000   620.317460     0.023007         0.001045   
...          ...          ...          ...              ...   
1267    0.000000   772.336051     0.027758         0.000405   
1268   27.584775   920.808450     0.026386         0.000619   
1269  354.483442  1077.393998     0.026299         0.000718   
1270  270.046015  1006.526309     0.027090         0.000551   
1271   31.143291  1234.263667     0.027540         0.001025   

      avg_speed_nearby_vehicles  
0                     65.350769  
1                     40.054214  
2                     44.091203  
3                     35.755170  
```

```
4                      41.627853
...                          ...
1267                   19.069430
1268                   27.404635
1269                   21.426673
1270                   29.590376
1271                   20.784419

[1272 rows x 47 columns]
```

In [48]:
```
result_New.fillna(value=0, inplace=True)  # Replaces NaN with 0
```

In [53]:
```
result_New['type'].value_counts()
```

Out[53]:
```
type
3    384
4    260
1    244
2    239
0    145
Name: count, dtype: int64
```

In [49]:
```
import pandas as pd

# Assuming df is your DataFrame
# Shuffle the rows using sample() function
shuffled_df = result_New.sample(frac=1, random_state=42)  # frac=1 shuffles th

# Reset the index if needed
shuffled_df.reset_index(drop=True, inplace=True)
```

Random Forest Evaluation

In [51]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Assuming 'df' is your DataFrame containing the data

# Features (X) and target variable (y)
X = shuffled_df.drop(['type','id'], axis=1)  # Features
y = shuffled_df['type']  # Target variable



# Predict on the test data
y_pred = clf_rand.predict(X)

# Calculate accuracy
accuracy = accuracy_score(y, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.7130503144654088
```

In [ ]: